

Corso : Tecnologie web AJAX Febbraio/Marzo/Aprile 2009

Prima Parte : Fondamenti di JAVASCRIPT

Prof. Sergio Cordella

2^ Lezione

Le strutture fondamentali

IF (Selezione semplice)

La forma completa **dell'istruzione IF** è la seguente:

```
if (espressione) {  
    istruzioni  
}  
  
else {  
    istruzioni  
}
```

Questo è un esempio molto semplice sull'utilizzo dell'istruzione IF.

```
if(text1 == "scuola")  
    alert("Hai scritto giusto!")  
else  
    alert("Attento! non hai scritto scuola!")
```

SE text1 (ovvero la casella di testo) è uguale alla parola casa, o più generalmente **SE** restituisce **VERO**, manda un messaggio "Hai scritto giusto" altrimenti manda un altro messaggio "Attento! non hai scritto casa".

Due ultime precisazioni prima di concludere con l'istruzione IF:

- se bisogna eseguire un **blocco di istruzioni** all'interno dell'IF occorre racchiuderle tra parentesi graffe. Esempio:

Quando si verifica la condizione tale che pippo sia maggiore di gigi voglio assegnare a pippo il valore di gigi e a gigi il valore 5 dovrò scrivere:

```
if (A > B) {  
    A = B  
    B = 5  
}
```

Se non avessi inserito le parentesi quando la condizione risulta vera (ovvero quando $A > B$) viene eseguita soltanto la prima istruzione ($A = B$) e la seconda viene eseguita sempre indipendentemente dalla condizione.

- in certi casi è possibile utilizzare più istruzioni **IF all'interno di altri IF**, ovvero vengono annidate più istruzioni (generalmente viene chiamata **ELSE IF**). Questa tecnica non è molto usata dai programmatori poichè rende il codice molto pesante e difficile da capire, quindi io vi consiglio di usare questa tecnica solo dove è necessario e sostituirla in caso con l'istruzione switch. Faccio subito un breve esempio:

```
if(a < 5)
    b = a
else
    if(c == b)
        c = 7
    else
        if(c > b)
            c = b
        else c = 0
```

Come ho già detto questa tecnica non è molto utilizzata per cui se non vi è chiara non preoccupatevi, la capirete più avanti quando diventeremo più bravi. Analizziamo ora questo esempio:

a < b ? a = c : b = c

Questa è un'istruzione **molto simile all' IF** infatti prima del segno ? si deve inserire necessariamente una espressione che restituisca un valore booleano (VERO o FALSO); se l'espressione è vera viene eseguito l'istruzione tra il carattere ? e : se invece è falsa viene eseguita l'istruzione successiva (quella dopo :).

SE a = 4 e b = 7 allora verrà assegnato il valore della variabile c alla variabile a. Non mi sembra molto difficile, comunque sarà tutto più facile e più chiaro quando avremmo finito questa parte del corso.

La stessa istruzione utilizzando l' IF sarebbe:

```
if(a < b)
    a = c
else b = c
```

E' evidente che utilizzando l'istruzione **?:** il programma diventa più leggibile.

SWITCH (selezione multipla)

Lo **SWITCH** è stato introdotto a partire da JavaScript1.2 ed è abbastanza difficile da spiegare a parole, comunque è molto simile all'istruzione ELSE IF descritta nella lezione precedente.

La forma completa dell'istruzione **SWITCH** è la seguente:

```
switch (espressione) {
    istruzioni
}
```

Vediamo subito un esempio pratico:

```
switch (numero) {
    case 1: /* se il numero è 1 esegue un'istruzione */; break;
    case 2: /* esegue un'altra istruzione */; break;
    case 3: /* ancora un'altra */; break;
    default: /* esegue l'istruzione di default */; break;
}
```

Ha una struttura più complessa rispetto all'istruzione IF ma comunque molto intuibile: se il numero è uguale ad 1 esegue le istruzioni presenti tra i due punti : e il break; (vedremo dopo a cosa serve). Si possono inserire più di una istruzione tra : e il break; però devono essere necessariamente separate da un punto e virgola.

Quando il numero non è uguale a nessun caso (in questo esempio ad 1, 2 e 3) esegue un'istruzione di default. Questa istruzione non è obbligatoria e quindi è possibile ometterla.

L'istruzione **break** fa in modo di uscire immediatamente da un ciclo più interno o da un'istruzione switch, infatti è molto utilizzato anche nei cicli (while - do while - for) che vedremo nelle prossime lezioni. La sua utilità sarà più chiara quando parleremo dei cicli.

Nello **switch** l'istruzione **break** serve per uscire appunto quando si è terminato di eseguire le istruzioni che lo precedono.

Nell'esempio se il numero è uguale a 2 e se il break fosse stato assente venivano eseguite prima le istruzioni dopo case 2: e dopo anche quelle riguardanti il case 3: e il default:.

Abbiamo acquisito nozioni sufficienti per realizzare una piccola calcolatrice che esegua le quattro operazioni principali (+ - * /) solo su due valori. Il problema è più semplice di quello che si possa credere infatti bisogna prima immettere i valori (utilizzando prompt()) e poi utilizzare un'istruzione switch per eseguire le operazioni. Il risultato può essere visualizzato sulla pagina web oppure utilizzando il metodo alert().

Codice JavaScript

```
<SCRIPT language="javascript1.2">
  var num1 = prompt("inserisci il primo numero","");
  var num2 = prompt("Inserisci il secondo numero","");
  var scelta = prompt("che cosa vuoi fare?","");

  switch(scelta) {
    case '+': ris = parseFloat(num1)+parseFloat(num2); break;
    case '-': ris = num1-num2; break;
    case '*': ris = num1*num2; break;
    case '/': ris = num1/num2; break;
    default: alert("scelta errata"); break;
  }
  alert(num1 + " " + scelta + " " + num2 + " = " + ris);
</SCRIPT>
```

Commenti :

num1 = prompt("inserisci il primo numero","")

Viene chiesto il primo numero utilizzando il metodo prompt() e il valore viene assegnato alla variabile num1 (vedi lezione 2); così anche per il secondo numero num2 e il simbolo dell'operazione da eseguire scelta.

case '+':

Quest'ultimo è un carattere e quindi scelta sarà una variabile stringa pertanto all'interno dell'istruzione SWITCH occorre inserire i caratteri delle operazioni che può assumere tra apici singoli.

ris = parseFloat(num1)+parseFloat(num2)

Se la variabile scelta ha una stringa di testo uguale a '+' si eseguono le istruzioni corrispondenti ovvero si sommano i due numeri e si salva il loro valore in un'altra variabile. Però bisogna stare attenti perchè invece di trattare num1 e num2 come variabili numeriche li stratta come stringhe, e quindi invece di sommare i due numeri li concatena. Perciò è stato necessario utilizzare la funzione parseFloat() in modo tale da renderli dei numeri. Vedremo più avanti come convertire stringhe in numeri e viceversa, l'importante è che abbiate capito l'istruzione **SWITCH**.

WHILE

In questa lezione oltre all'istruzione **WHILE** parleremo anche dei contatori e del loro utilizzo poichè vengono utilizzati all'interno dei cicli.

Un contatore prima di tutto va dichiarato e inizializzato (gli si assegna un valore numerico):

nome_contatore = valore

Generalmente il contatore si inizializza a zero ma può capitare di inizializzarlo a 1 o più generalmente ad altri valori. Il contatore viene utilizzato molto spesso e serve per contare appunto quante volte viene ripetuto il ciclo. Il contatore va incrementato ovvero deve aumentare il suo valore di una unità ogni volta che si ripete il ciclo:

```
contatore=0
inizio ciclo
    contatore++
fine ciclo
```

Scrivere contatore++ equivale a scrivere contatore=contatore+1. E' possibile anche decrementare il contatore: contatore--. Passiamo ora all'istruzione **WHILE**.

L'istruzione **WHILE** (o controllo in testa) e **DO WHILE** (o controllo in coda) vengono utilizzati per creare dei cicli iterativi, ovvero iterano un'istruzione.

La forma completa dell'istruzione **WHILE** è la seguente:

```
while (espressione) {
    istruzioni
}
```

Il ciclo **WHILE** esegue l'istruzione (o le istruzioni) **finchè l'espressione non risulta falsa**.

Per esempio:

```
<SCRIPT language="javascript">
    var contatore = 1;
    var n = prompt("inserisci un numero","");
    while (n != 5) {
        n = prompt("inserisci un numero","");
        contatore++;
    }
    alert("finalmente hai inserito 5!");
    alert("hai inserito " + contatore + " numeri");
</SCRIPT>
```

- viene chiesto di inserire la prima volta il numero n poi si entra nel ciclo
- controlla se n è diverso da 5
- se è diverso (quindi se la condizione è vera) richiede un'altra volta il numero
- esegue l'istruzione racchiusa tra parentesi graffe finchè n è uguale a 5.
- il contatore viene incrementato ogni volta che si chiede un numero.
- esce dal ciclo e comunica "finalmente hai inserito 5!" e quanti numeri hai inserito.

E' stato necessario inizializzare il contatore a 1 poichè il numero viene chiesto una volta fuori dal ciclo.

DO WHILE

Funziona esattamente come WHILE solo che l'espressione viene controllata alla fine del ciclo anzichè all'inizio. Ciò significa che il ciclo viene eseguito almeno una volta. La sintassi è:

```
do {  
    istruzioni  
} while (espressione)
```

Per esempio proviamo ad applicarlo all'esempio precedente:

```
<SCRIPT language="javascript">  
    var contatore = 1;  
    var n = prompt("inserisci un numero","");  
  
    do {  
        n = prompt("inserisci un numero","");  
        contatore++;  
    } while(n != 5);  
  
    alert("finalmente hai inserito 5!");  
    alert("hai inserito " + contatore + " numeri");  
</SCRIPT>
```

Si pensa che siano uguali i due cicli poichè non accade niente di diverso nei due esempi ma se proviamo ad inserire come **primo numero 5** vedremo che non è così. Infatti nel primo esempio **prima del ciclo controlla se la condizione è falsa e quindi esce subito senza chiedere un altro numero; invece nel ciclo DO WHILE la condizione viene controllata a partire dal secondo numero.**

Cerchiamo di migliorare la calcolatrice realizzata nella precedente aggiungendo un ciclo DO WHILE e qualche ritocco.

```
<SCRIPT language="javascript">  
    var num1 = prompt("inserisci il primo numero","");  
    var num2 = prompt("Inserisci il secondo numero","");  
    do {  
alert("0. termina\n1. cambia numeri\n2. somma\n3. sottrai\n4. moltiplica\n5. dividi");  
        var scelta = prompt("che cosa vuoi fare?","");  
        switch(scelta) {  
case '1': num1 = prompt("inserisci il primo numero","");  
            num2 = prompt("Inserisci il secondo numero","");  
            break;  
case '2': alert(num1 + " + " + num2 + " = " + (parseFloat(num1)+parseFloat(num2)));break;  
case '3': alert(num1 + " - " + num2 + " = " + (num1-num2)); break;  
case '4': alert(num1 + " * " + num2 + " = " + (num1*num2)); break;  
case '5': alert(num1 + " / " + num2 + " = " + (num1/num2)); break;  
            default: alert("scelta errata"); break;  
        }  
    } while(scelta != 0);  
    alert("fine programma!");  
</SCRIPT>
```

Lo script continua finchè non viene inserito il numero zero, ovvero finche la condizione del ciclo do-while risulta vera (scelta != 0); quando la variabile scelta ha valore zero allora lo script termina.

E' molto facile trovare, soprattutto nei programmi realizzati da "non principianti", la seguente forma:

```
...
while (scelta);
...
che è equivalente a:
...
while (scelta != 0);
...
```

FOR

Il **ciclo FOR** è simile al ciclo WHILE; serve per ripetere una o più istruzioni un determinato numero di volte. La sintassi è la seguente:

```
for(espressione_iniziale; condizione; incremento) {
    istruzioni
}
```

Cosa accade all' interno di un ciclo for?

1. Viene eseguita l' espressione iniziale **espressione_iniziale**, se e' presente (questa espressione di solito e' usata per inizializzare uno o piu' contatori).
2. Viene **valutata la condizione**. Se risulta vera, le istruzioni del ciclo vengono eseguite, altrimenti il ciclo termina.
3. Viene **eseguito l'incremento** (questa espressione di solito e' usata per incrementare uno o piu' contatori).
4. Vengono eseguite le istruzioni, ed il controllo ritorna al punto 2.

Ecco un esempio:

```
<SCRIPT language="javascript">
    for(i = 1; i <= 5; i++)
        alert(i);
</SCRIPT>
```

Questo esempio ripete 5 volte la stessa istruzione incrementando il contatore sempre di una unità (i++).

Quando bisogna ripetere una sola istruzione non è necessario racchiudere l'istruzione tra le parentesi graffe. Questo vale per tutte le istruzioni di controllo quali: IF ELSE - SWITCH - WHILE / DO WHILE - FOR.

Scrivere:

```
for(i = 1; i <= 5; i++)
    alert(i)
```

equivale a scrivere:

```
for(i = 1; i <= 5; i++) {  
    alert(i)  
}
```

Esercizio : Calcolare la media di una sequenza di numeri.

Dovremmo dire quanti numeri dobbiamo inserire e poi inserirli uno per volta, utilizzare il ciclo FOR, calcolare la media e comunicare il risultato. E' più difficile di quello che sembra, vedrete che in pochi minuti riuscirete a realizzarlo anche voi. Incominciamo!!!

JavaScript

```
<SCRIPT language="javascript">  
    var sommatore = 0;  
    var sequenza = prompt("quanti numeri vuoi inserire?", "");  
    sequenza = parseFloat(sequenza);  
  
    for(i=0;i<sequenza;i++) {  
        numero = prompt("inserisci il numero" + i, "");  
        sommatore += parseFloat(numero);  
    }  
    var media = sommatore/sequenza;  
    alert("la media è: " + media);  
</SCRIPT>
```

L'unica novità introdotta in questo esempio è l'utilizzo del sommatore. Va dichiarato e inizializzato e viene utilizzato all'interno di un ciclo per sommare più numeri in una stessa variabile:

```
    sommatore=0  
        inizio ciclo  
            sommatore += numero  
        fine ciclo
```

Scrivere `sommatore += numero` equivale a scrivere `sommatore = sommatore + numero`.

Oltre al sommatore esistono:

```
variabile -= numero --> variabile = variabile - numero  
variabile *= numero --> variabile = variabile * numero  
variabile /= numero --> variabile = variabile / numero  
variabile %= numero --> variabile = variabile % numero
```

Ricordatevi sempre che sia il contatore che il sommatore vanno inizializzati; il perchè è evidente: se io non so che valore assume il sommatore non posso sommarci se stesso più un altro numero.

Ora creeremo un banalissimo programma per **calcolare il fattoriale di un numero**.

```
<SCRIPT language="javascript">  
    var fatt = 1;  
    var num = prompt("inserisci il numero", "");  
    for(i = 1; i <= num; i++)  
        fatt *= i;  
    alert(num + "!" + fatt);  
</SCRIPT>
```

E' possibile ridurre al minimo il codice inserendone una parte all'interno delle parentesi del for.
Vediamo come:

```
<SCRIPT language="javascript">
  var num = prompt("inserisci il numero","");
  for(i = 1, fatt = 1; i < num; i++, fatt *= i) ;
  alert(num + "! =" + fatt);
</SCRIPT>
```

Bisogna fare **attenzione di inserire il punto e virgola dopo il for altrimenti ripeterebbe il metodo alert() ogni volta che si ripete il ciclo**. Comunque questo sistema non è molto usato però è bene conoscerlo.

Ora vediamo come è possibile generare **un ciclo (loop)** infinito utilizzando l'istruzione for:

```
for(;;) {
  istruzioni
}
```

Anche questo utilizzo dell'istruzione for non è usata molto spesso; se non riuscite a cogliere la sua utilità ve ne accorgete più avanti...

Io mi fermerei qui senza approfondire l'argomento e creare un po' di confusione, l'importante è che abbiate capito il ciclo **for** e tutto il capitolo 1 in generale (strutture di controllo) perchè sono la base fondamentale di un linguaggio di programmazione.

Solo un'ultima precisazione su un'altra istruzione utilizzata all'interno di cicli simile al **break**. Ora farò un piccolo accenno dell'istruzione **continue** che permette di ripetere un ciclo (**e quindi saltare le istruzioni sottostanti**) quando una condizione risulta vera. Analizziamo questo esempio:

```
for (i = 0; i < 10; i++) {
  a = b+c
  if(a == 3)
    continue
  b = c+a
  c++
}
```

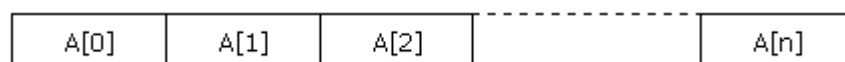
Se a è uguale a 3 vengono saltate le istruzioni sottostanti a continue e viene ripreso il ciclo.

L'istruzione **continue** può essere utilizzata all'interno di **cicli for, while e do/while** ma **non** all' interno di strutture di **selezione** (quali **IF ELSE**).

I VETTORI o ARRAY

In questo capitolo parleremo degli **Array** (dall'inglese il termine array significa qualcosa di simile a "disposizione ordinata di elementi di natura omogenea"). E' la struttura dati più utilizzata in molti linguaggi di programmazione e in qualche caso (come nei linguaggi macchina) è l'unico strumento a disposizione del programmatore per memorizzare ed elaborare aggregati di dati.

Un array (o anche detto vettore) è un contenitore idealmente costituito da tante variabili semplici individuabili tramite un indice intero. L'immagine sottostante rappresenta la schematizzazione di un array.



Ogni cella dell'array è una semplice variabile. A prima vista l'utilità di questa struttura può sembrare molto ridotta invece come dicevo prima è una delle più utilizzate! La memoria di un computer tradizionale è organizzata come un array e ciò spiega la grande popolarità di questo strumento: ogni vettore utilizzato in un programma occupa generalmente una certa porzione di celle contigue della memoria del computer.

Passiamo ora alla **dichiarazione di un array**:

```
var A = new Array();
```

A differenza di molti linguaggi di programmazione (soprattutto di medio-basso livello come il C) il JavaScript **permette una allocazione dinamica della memoria implicita**: l'interprete JavaScript si accorge se occorre utilizzare (allocare) della memoria per memorizzare nuovi dati e gestisce queste operazioni autonomamente senza che il programmatore si accorga di nulla.

Questo permette al programmatore di non impostare alcun limiti alla lunghezza del vettore.

A questo punto mi sembra abbastanza chiaro che la lunghezza del vettore è rappresentata dal numero di elementi (variabili) dell'array.

Il secondo modo di richiamare il costruttore Array() consente di specificare esplicitamente valori per i primi n elementi di un array:

```
var A = new Array(5, 6, 9, 3, "test", 8, false, 1);
```

Il terzo modo per richiamare il costruttore Array() è quello di utilizzare un singolo elemento numerico che specifica la lunghezza del vettore:

```
var A = new Array(12);
```

Questa tecnica crea un array con il numero specificato di elementi (ciascuno dei quali ha valore undefined).

Con JavaScript 1.2 sono stati introdotti i **letterali array**, che mettono a disposizione un ulteriore modo per creare array. Un letterale array consente di incorporare un valore di array direttamente in un programma JavaScript nello stesso modo si definisce una qualsiasi variabile.

Per creare un **letterale array** è sufficiente inserire un elenco di valori separati da virgole tra parentesi quadre come negli esempi seguenti:

```
A = [5, 6, 9, 3, "test", 8, false, 1, 'c'];
```

```
vett = [7, 3.14, true, false, false, "ok, ok!"];
```

Come avrete sicuramente notato **gli array possono** contenere (come del resto le singole variabili) **elementi di tipo diverso**. Per esempio nell'esempio precedente l'array A contiene elementi di tipo intero, stringa, booleano e carattere.

Per accedere agli elementi di un array si utilizza **l'operatore []**. A sinistra delle parentesi deve comparire un riferimento all'array. All'interno delle parentesi deve trovarsi un'espressione arbitraria di valore intero positivo (un indice). Tale sintassi può essere utilizzata sia per leggere che per scrivere un valore in un elemento di un array.

Tutti gli esempi seguenti sono corretti:

```
var valore;
```

```
var i = 2;
```

```
var A = new Array();
```

```
A[0] = true;
```

```
valore = A[0];
```

```
A[1] = 3.14;
```

```
A[i] = 4;
```

```
A[i + 1] = "ciao";
```

```
A[A[i]] = A[0];
```

Ora le variabili e l'array conterranno questi valori:

true	3.14	2	"ciao"	true
A[0]	A[1]	A[2]	A[3]	A[4]

2	true
i	valore

In molti linguaggi di programmazione il primo elemento di un array è rappresentato dall'indice 1 (per esempio il Pascal), mentre **in JavaScript (come in C, C++ e Java) il primo elemento ha indice 0**.

Come già accennato nella lezione precedente nei linguaggi come C e Java, un array ha un numero fisso di elementi che devono essere specificati al momento della sua creazione. Ciò non avviene in JavaScript, dove un array può avere un qualsiasi numero di elementi e tale numero può essere modificato in qualsiasi momento.

Gli array in JavaScript sono frammentati. Ciò significa che gli indici degli array non devono appartenere ad un intervallo continuo di numeri.

La memoria viene allocata solo per gli elementi dell'array effettivamente contenuti.

Come le seguenti righe di codice, l'interprete JavaScript alloca memoria solo per gli indici 0 e 10000 dell'array **e non per i 9999 indici tra essi compresi**.

```
A[0] = "il primo elemento";
```

```
A[10000] = "elemento 10000";
```

Tutti gli array hanno una **proprietà length** che specifica il numero di elementi contenuti nell'array. Il suo utilizzo è molto semplice; vediamo alcuni esempi:

```
var A = new Array();  
//A.length == 0  
//nessun elemento definito
```

```
var B = new Array(12);  
//B.length == 12  
//definiti elementi vuoti 0-11
```

```
var C = new Array(1,3,6,8);  
//C.length == 4  
//definiti elementi 0-3
```

```
var D = [4,6];  
//D.length == 2  
//definiti elementi 0 e 1
```

```
var E = new Array(5,2,7);  
E[99] = 1;  
//E.length == 4  
//definiti elementi 0, 1, 2 e 99
```

Una applicazione pratica usatissima di questa proprietà è la seguente. Il ciclo for permette di scorrere il vettore utilizzando l'indice i.

```
var sport = ["calcio", "golf", "tennis", "hokey", "basket"];
for(i = 0; i < sport.length; i++)
    alert(sport[i]);
```

L'esempio precedente assume che gli elementi dell'array siano tutti contigui, ovvero che siano stati tutti utilizzati, e che abbiano inizio dall'indice 0. In caso contrario bisogna verificare che ciascun elemento dell'array sia stato definito prima di poterlo utilizzare.

Vi propongo due esempi: il primo visualizza tutti gli elementi (anche quelli non dichiarati) mentre il secondo effettua un controllo.

```
var sport = ["calcio", "golf"];
sport[3] = "tennis";
sport[6] = "hokey";
for(i = 0; i < sport.length; i++)
    alert(sport[i]);
```

```
var sport = ["calcio", "golf"];
sport[3] = "tennis";
sport[6] = "hokey";
for(i = 0; i < sport.length; i++)
    if(sport[i]) alert(sport[i]);
```

Metodi disponibili per l' Array

Per gestire gli array sono disponibili, oltre all'operatore [], numerosi metodi messi a disposizione dalla classe Array.

Metodo	Descrizione
concat()	Restituisce un array risultato di due o più array concatenati Esempio: array_unione = array1.concat(array2)
join()	Unisce gli elementi di un array in una stringa con i valori divisi da un separatore (il default è la virgola) Esempio: <script> var nomi = new Array("Pippo", "Nonna Papera", "Paperino", "Topolino", "Minnie", "Qui") var stringa = nomi.join(";"); <u>document</u> .write(stringa); </script> Output: Pippo;Nonna Papera;Paperino;Topolino;Minnie;Qui

pop() Restituisce l'ultimo elemento dell'array e lo rimuove dall'array

Esempio:

```
<script>
var nomi = new Array("Pippo", "Nonna
Papera", "Paperino", "Topolino", "Minnie", "Qui")
var stringa = nomi.join(";");
document.write("Prima del pop: "+stringa+"<br>");
document.write("Ultimo elemento: "+nomi.pop()+"<br>");
var stringa = nomi.join(";");
document.write("Dopo il pop: "+stringa);
</script>
```

Output:

Prima del pop: Pippo;Nonna Papera;Paperino;Topolino;Minnie;Qui

Ultimo elemento: Qui

Dopo il pop: Pippo;Nonna Papera;Paperino;Topolino;Minnie

push() Aggiunge uno o più elementi all'array e ne restituisce la nuova lunghezza

Esempio:

```
<script>
var nomi = new Array("Pippo", "Nonna
Papera", "Paperino", "Topolino", "Minnie", "Qui")
var stringa = nomi.join(";");
document.write("Prima del push: "+stringa+"<br>");
document.write("Lunghezza: "+nomi.length+"<br>");
document.write("Lunghezza dopo il push: "+nomi.push("Quo", "Qua")+"<br>");
var stringa = nomi.join(";");
document.write("Dopo il push: "+stringa);
</script>
```

Output:

Prima del push: Pippo;Nonna Papera;Paperino;Topolino;Minnie;Qui

Lunghezza: 6

Lunghezza dopo il push: 8

Dopo il push: Pippo;Nonna Papera;Paperino;Topolino;Minnie;Qui;Quo;Qua

reverse() **Inverte l'ordine degli elementi dell'array**

Esempio:

```
<script>
var nomi = new Array("Pippo", "Nonna
Papera", "Paperino", "Topolino", "Minnie", "Qui")
var stringa = nomi.join(";");
document.write("Prima del reverse: "+stringa+"<br>");
nomi.reverse();
var stringa = nomi.join(";");
document.write("Dopo il reverse: "+stringa);
</script>
```

Output:

Prima del reverse: Pippo;Nonna Papera;Paperino;Topolino;Minnie;Qui

Dopo il reverse: Qui;Minnie;Topolino;Paperino;Nonna Papera;Pippo

shift() **Rimuove e restituisce il primo elemento dell'array**

Esempio:

```
<script>
```

```
var nomi = new Array("Pippo","Nonna
```

```
Papera","Paperino","Topolino","Minnie","Qui")
```

```
var stringa = nomi.join(";");
```

```
document.write("Prima dello shift: "+stringa+"<br>");
```

```
document.write("Primo elemento: "+nomi.shift()+"<br>");
```

```
var stringa = nomi.join(";");document.write("Dopo lo shift: "+stringa);
```

```
</script>
```

Output:

Prima dello shift: Pippo;Nonna Papera;Paperino;Topolino;Minnie;Qui

Primo elemento:

Pippo

Dopo lo shift: Nonna Papera;Paperino;Topolino;Minnie;Qui

slice() **Crea un nuovo array a partire da una porzione dell'array originale**

Esempio:

```
<script>
```

```
var nomi = new Array("Pippo","Nonna
```

```
Papera","Paperino","Topolino","Minnie","Qui")
```

```
var stringa = nomi.join(";");
```

```
document.write("Prima dello slice: "+stringa+"<br>");
```

```
nomi = nomi.slice(3);var stringa = nomi.join(";");
```

```
document.write("Dopo lo slice: "+stringa);
```

```
</script>
```

Output:

Prima dello slice: Pippo;Nonna Papera;Paperino;Topolino;Minnie;Qui

Dopo lo slice: Topolino;Minnie;Qui

sort() **Ordina gli elementi dell'array**

Esempio:

```
<script>
```

```
var nomi = new Array("Pippo","Nonna
```

```
Papera","Paperino","Topolino","Minnie","Qui")
```

```
var stringa = nomi.join(";");
```

```
document.write("Prima del sort: "+stringa+"<br>");
```

```
nomi.sort();
```

```
var stringa = nomi.join(";");
```

```
document.write("Dopo il sort: "+stringa);
```

```
</script>
```

Output:

Prima del sort: Pippo;Nonna Papera;Paperino;Topolino;Minnie;Qui

Dopo il sort: Minnie;Nonna Papera;Paperino;Pippo;Qui;Topolino

toString() Restituisce una stringa che rappresenta gli elementi dell'array

Esempio:

```
<script>
```

```
var nomi = new Array("Pippo","Nonna  
Papera","Paperino","Topolino","Minnie","Qui")
```

```
nomi.toString();
```

```
document.write(nomi);
```

```
</script>
```

Output:

Pippo,Nonna Papera,Paperino,Topolino,Minnie,Qui

unshift() Aggiunge uno o più elementi all'inizio di un array e ne restituisce la nuova lunghezza

Esempio:

```
<script>
```

```
var nomi = new Array("Pippo","Nonna  
Papera","Paperino","Topolino","Minnie","Qui")
```

```
var stringa = nomi.join(";");
```

```
document.write("Prima dell'unshift: "+stringa+"<br>");
```

```
nomi.unshift("Quo","Qua");
```

```
var stringa = nomi.join(";");
```

```
document.write("Dopo l'unshift: "+stringa);
```

```
</script>
```

Output:

Prima dell'unshift: Pippo;Nonna Papera;Paperino;Topolino;Minnie;Qui

Dopo l'unshift: Quo;Qua;Pippo;Nonna Papera;Paperino;Topolino;Minnie;Qui

MATRICI

Parleremo velocemente degli **array multidimensionali** o **matrici**, specialmente quelli a due dimensioni che vengono anche chiamati matrici. Una matrice si presenta nel seguente schema:

	0	1	2	...	N
0					
1					
2		A[2][1]			
...					
N					

Come potete notare la matrice è formata da N vettori, quindi per accedere ad un elemento della matrice ci vogliono due indici: il primo identifica la riga mentre il secondo identifica la colonna! Vediamo con alcuni esempi vari modi per dichiarare una matrice.

Il primo esempio utilizza il costruttore Array(), creando prima un vettore e poi per ogni elemento di esso, tramite un ciclo for, altri vettori annidati creando appunto una matrice.

```
var nRighe = 3;  
var nColonne = 5;  
var a = new Array(nRighe);  
  
for(i = 0; i < nRighe; i++)  
  a[i] = new Array(nColonne);
```

In questo modo abbiamo creato una matrice **3x5**, ovvero con 3 righe e 5 colonne.

Esiste un'altro modo per creare una matrice, utilizzando i literal array:

```
var a = [[0,0,0,0,0], [0,0,0,0,0], [0,0,0,0,0]];
```

E' stata creata una matrice **3x5** con tutti gli elementi inizializzati a zero.

Vediamo, con un esempio, come è possibile accedere agli elementi della matrice:

```
var nRighe = 2;  
var nColonne = 4;  
var a = new Array(nRighe);  
  
for(i = 0; i < nRighe; i++)  
  a[i] = new Array(nColonne);  
  
for(i = 0; i < nRighe; i++)  
  for(j = 0; j < nColonne; j++)  
    a[i][j] = prompt("inserisci elemento in riga "+ i +" e colonna "+ j +":", "");  
  
for(i = 0; i < nRighe; i++)  
  for(j = 0; j < nColonne; j++)  
    alert("elemento in riga "+ i +" e colonna "+ j +": " + a[i][j]);
```

Le matrici vengono usate molto spesso per risolvere problemi come la battaglia navale appunto perchè la matrice deve gestire una tabella di celle del campo di gioco.